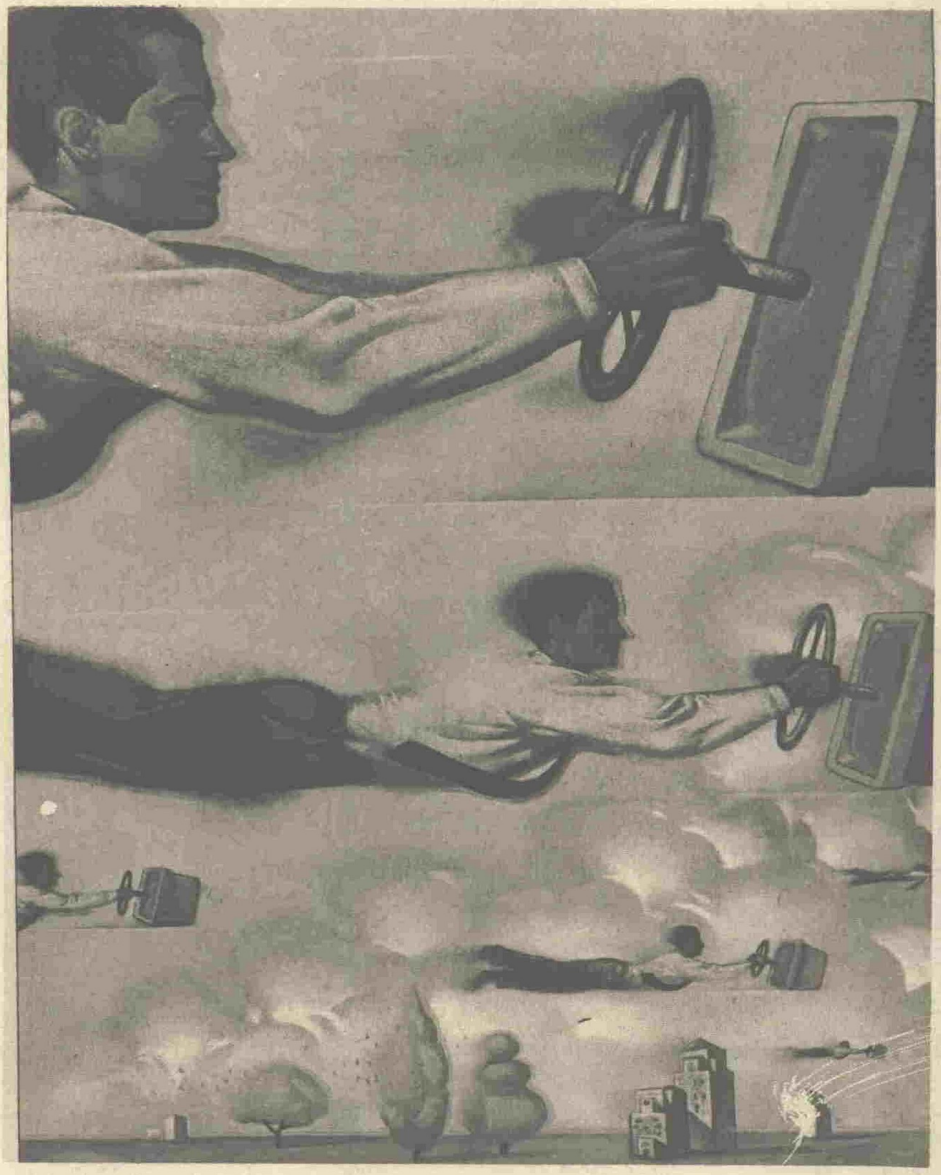
ยืน ภู่วรวรรณ

มั่นใจกว่ากับการจัดการข้อมูล X-Base



มองให้ลึกซึ้งจึงเห็นเทคนิคเล็ก ๆ น้อย ๆ กับการจัดการฐานข้อมูล

การจัดการข้อมูลที่ใช้กันมากในเมืองไทยขณะนี้เห็นจะได้แก่ การใช้ดีเบส ฟอกซ์เบส และคลิปเปอร์ ทั้งสามตัวนี้มีลักษณะคำสั่งของการเขียนโปรแกรมคล้ายกัน จนทำให้โปรแกรมที่เขียนด้วยภาษาที่ใช้กับซอฟต์แวร์ทั้งสามตัวนี้ใช้งานร่วมกันได้ ถึงแม้ในขณะนี้จะมีการพัฒนาเลยออกไปกันมากแล้วก็ตาม พื้นฐานคำสั่งทั่วไปก็ยังคงเหมือนกัน และด้วยปัญหาที่พัฒนาการเริ่มแตกแยกออกไปคนละทิศละทาง ซึ่งอาจสร้างปัญหาได้ในอนาคต จึงมีกลุ่มบุคคลในสหรัฐอเมริกาพยายามสร้างองค์กรและดึงบริษัทผู้ผลิตซอฟต์แวร์ดาต้าเบสที่มีรากฐานในกลุ่มนี้มาสร้างมาตรฐานกลางร่วมกัน และตั้งชื่อว่า X-Base จึงเป็นสิ่งที่เมื่อมีการกล่าวถึงก็จะใช้ได้กับซอฟต์แวร์หลาย ๆ ตัว เป็นมาตรฐานกลางเหมือนการเรียกฟอร์แทรน ปาสคาล แนวโน้มของ X-Base จะเป็นอย่างไร ก็คงจะต้องติดตามดูกันต่อไป

สำหรับผู้ที่ใช้งานบน X-Base ทั่วไปมักพบปัญหาอย่างหนึ่งคือ เมื่อแฟ้มข้อมูลมีขนาดใหญ่ขึ้นหรือมีการสร้างแฟ้มข้อมูลหลายแฟ้มที่เชื่อมโยงกัน เมื่อทำการปรับปรุงข้อมูลจะพบปัญหาหลายอย่าง ในฉบับนี้จึงขอนำเทคนิคเล็ก ๆ น้อย ๆ เกี่ยวกับการจัดการข้อมูล

ปัญหาที่เกิดจาก DELETE และ PACK

คำสั่ง DELETE เป็นคำสั่งที่เข้าใจง่าย เพราะใช้สำหรับการลบข้อมูลเรคอร์ดที่กำหนดออกไป โดยความจริงแล้วข้อมูลที่ลบยังคงอยู่ แต่ X-Base ทำการกำหนดสถานะของเรคอร์ดนั้นซึ่งเป็นแฟล็กบอกสถานะการลบออกแล้วเป็น "T" หากใช้คำสั่ง REC-COUNT ( ) ดูจะพบว่าจำนวนเรคอร์ดที่อยู่ในไฟล์ยังคงเท่าเดิมไม่มีการเปลี่ยนแปลง แต่หากใช้คำสั่ง LIST, SKIP จะไม่พบเรคอร์ดที่ลบนั้น

หากต้องการเข้าถึงเรคอร์ดที่ลบ เราทำได้ด้วยการใช้คำสั่ง SET DELETE OFF ข้อมูล ส่วนเรคอร์ดที่ลบแล้วจะยังคงเหมือนข้อมูลเรคอร์ดอื่น ซึ่งสามารถ list หรือเข้าถึงได้อย่างปกติ การลบแฟล็กหรือเรียกข้อมูลเรคอร์ดที่ลบคืนก็ใช้คำสั่ง RECALL

ในความต้องการของผู้ใช้อยากจะลบเรคอร์ดออกจริง ๆ โดยให้เรคอร์ดที่ลบไปเก็บไว้ในไฟล์สำรอง การเรียกคืนก็ให้เรียกจากไฟล์นั้นมาใส่ แต่ความเป็นจริงของระบบการจัดการข้อมูลบนแผ่นดิสก์เปลี่ยนแปลงข้อมูลเช่นนั้นเป็นเรื่องยุ่งยาก ดังนั้นการลบจึงเป็นเพียงการกำหนดแฟล็กให้เป็นจริงเท่านั้น

หากต้องการลบจริงต้องใช้คำสั่ง PACK คำสั่ง PACK นี้จะทำการคัดลอกข้อมูลจากแฟ้มเดิมไปใส่ในแฟ้มใหม่ โดยคัดเอาเฉพาะเรคอร์ดที่แฟล็กการลบไม่เป็นจริง เมื่อคัดลอกครบแล้วจะเปลี่ยนชื่อแฟ้มเป็นชื่อเดิมแล้วลบแฟ้มเดิมทิ้งไป

ปัญหาที่เกิดขึ้นประการแรกคือการ PACK ต้องการเนื้อที่เพิ่มขึ้นอีกเท่าตัว ดังนั้นผู้เขียนโปรแกรมจัดการข้อมูลต้องคำนึงข้อนี้ด้วย หากที่ไม่พอ X-Base จะถามโดยอาจให้ลบแฟ้มอื่นออกแล้วทำต่อผู้ใช้งานต้องระมัดระวังเป็นพิเศษ

ปัญหาประการต่อมา ที่ผู้เขียนโปรแกรมไม่ระมัดระวังคือ เรื่องแฟ้มดัชนี การลบเรคอร์ดออกจะทำให้แฟ้มดัชนีมีปัญหา สมมติว่าแฟ้มที่กำลังทำงานมีแฟ้มดัชนีประกอบอยู่หลายแฟ้ม การ PACK จะทำให้เรคอร์ดบนแฟ้มหลักหายไป แต่ถ้าเพิ่มดัชนีเปิดร่วมอยู่ด้วยก็จะจัดการแฟ้มดัชนีนั้นด้วย แต่ถ้าแฟ้มดัชนีไม่เปิดร่วมก็ต้องกลับไปทำคำสั่ง REINDEX เพื่อจัดการกับแฟ้มดัชนีที่เหลือ

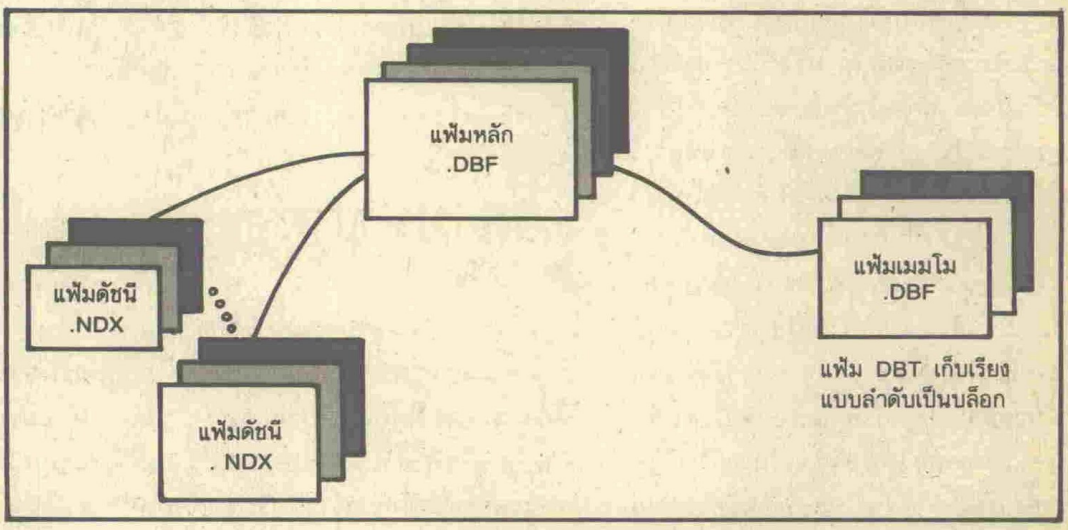
PACK กับฟิล์ดเมมโม

ก่อนอื่นมาดูโครงสร้างของแฟ้มที่เกาะเชื่อมอยู่ดังรูปที่ 1 แฟ้มเมมโมเป็นแฟ้มเอกสารที่เกาะเชื่อมโดยเก็บเป็นบล็อก ๆ ละ 512 ไบต์ ส่วนของพอยน์เตอร์บนแฟ้มหลังจะชี้บอกว่าเชื่อมกับบล็อกใดในแฟ้ม .dbt

การ PACK จะไม่มีผลต่อการลบข้อมูลในส่วน .dbt เลย ข้อมูลในแฟ้ม .dbt ยังคงอยู่อย่างเดิมตามปกติ ดังนั้นเมื่อไม่มีตัวชี้จากแฟ้มหลักมายัง .dbt ข้อมูลส่วนบล็อกนั้นก็จะกลายเป็นขยะ ทางแก้ไขที่ทำได้คือใช้คำสั่ง COPY จะเป็นผลทำให้เกิดการสร้างแฟ้ม .dbt ตัวใหม่ที่ตัดบล็อกที่ไม่ใช้ออกไป

หลีกเลี่ยงการใช้ PACK

หากฐานข้อมูลมีขนาดใหญ่และมีการเชื่อมกับหลายแฟ้มในลักษณะที่มีแฟ้มดัชนีเกาะติดอยู่รวมทั้งแฟ้ม .dbt



รูปที่ 1 โครงสร้างเชื่อมโยงระหว่างแฟ้ม

ด้วย การใช้ PACK จึงเป็นการสร้างปัญหา เพราะจะมีผลต่อดัชนีทุกแฟ้ม ผู้เขียนโปรแกรมต้องคำนึงไว้เป็นกรณีพิเศษ

การลบข้อมูลจึงไม่ควรยุ่งเกี่ยวกับการเปลี่ยนแปลงลดขนาด เพราะแฟ้มที่อยู่บนเอ็มเอสดอสบนฮาร์ดดิสก์ เมื่อเก็บข้อมูลเป็นจำนวนมากขนาดของแฟ้มถ้ามีการย่นเรคอร์ดใดเรคอร์ดหนึ่งเข้ามาจะต้องเกี่ยวพันกับการเขียนและการอ่านข้อมูลจำนวนมากทันที ขณะการเขียนหรืออ่านข้อมูลจำนวนมากอาจสร้างปัญหาโดยเฉพาะแฟ้มดัชนีที่เชื่อมโยงอยู่ต้องเสียเวลามากในการจัดการทำดัชนีใหม่

ประสานการลบกับการเพิ่มข้อมูลเข้าด้วยกัน

แฟ้มข้อมูลที่จัดเก็บ อาจมีการเรียกข้อมูลบางเรคอร์ดมาแก้ไข เปลี่ยนแปลง เช่น การใช้คำสั่ง CHANGE, EDIT BROWSE หรือในโปรแกรมมีการใช้คำสั่ง GET และ READ สำหรับการเปลี่ยนแปลงแก้ไขข้อมูล

อย่างไรก็ดี ถ้ามีการเพิ่มข้อมูลในแฟ้ม การเพิ่มส่วนใหญ่เป็นการ APPEND คือการใส่เรคอร์ดเพิ่มต่อท้าย การเพิ่มเรคอร์ดต่อท้ายจึงเป็นเรื่องที่ไม่มีปัญหา แต่หากแทรกข้อมูลระหว่างกลางจะมีปัญหาต่อการเขียนอ่านข้อมูลจำนวนมาก ดังนั้นจึงให้หลีกเลี่ยงคำสั่ง INSERT แต่ใช้เพิ่มข้อมูลต่อท้าย

เพื่อเพิ่มประสิทธิภาพโดยรวมของระบบให้ดียิ่งขึ้น จึงให้ใช้การเพิ่มและลบข้อมูลที่ประสานกัน กล่าวคือ เมื่อมีการลบข้อมูลก็ทำเครื่องหมายหรือสถานะแฟล็กให้ลบไว้ ในกรณีนี้ก็ใช้ DELETE ได้ หรือจะทำเครื่องหมาย

บางอย่างไว้ เช่น ทำข้อมูลให้เป็นช่องว่าง แต่เมื่อต้องการจะเพิ่มข้อมูลก็ค้นหาเรคอร์ดเหล่านี้แล้วเปลี่ยนแปลงข้อมูลดังกล่าวให้เป็นข้อมูลเรคอร์ดที่จะเพิ่ม หากไม่มีเรคอร์ดที่ลบไว้ก่อนจึงค่อยเพิ่มต่อท้ายแฟ้ม การค้นหาเรคอร์ดที่ลบแล้วมีการตรวจสอบด้วยฟังก์ชัน DELETE ( ) ได้

วิธีการเพิ่มเรคอร์ดเข้าแทนที่เรคอร์ดที่ลบนี้ให้ข้อดีในส่วนที่ระบบจะไม่ยุ่งเกี่ยวกับโครงสร้างความยาวของแฟ้ม การเขียนและอ่านจะทำเฉพาะส่วนบริเวณเรคอร์ดที่ลบเพิ่มเท่านั้น วิธีการนี้จะมีผลต่อข้อมูลในแฟ้มดัชนีที่เปิดไว้ สิ่งที่ระมัดระวังคือ ข้อมูลที่อยู่ในแฟ้มดัชนีที่ไม่เปิดจะต้องทำการ RE-INDEX และหากเชื่อมโยงเข้ากับ .dbt ต้องระมัดระวังเพราะ .dbt ที่ลบนี้ยังคงอยู่

ระวังการใช้ APPEND BLANK

การใช้ APPEND BLANK เป็นการเพิ่มเรคอร์ดที่ว่าง ๆ เข้าไปในแฟ้มหนึ่งเรคอร์ด มีผู้เขียนโปรแกรมจำนวนมากนิยมใช้คำสั่ง APPEND BLANK เพิ่มเรคอร์ดว่างแล้วสร้างสกรีนรับข้อมูลใหม่ใส่ลงในเรคอร์ดว่างนี้ กล่าวโดยตรงคือการให้ผู้ป้อนข้อมูลใส่ข้อมูลลงในเรคอร์ดว่างโดยตรงนั่นเอง

โดยปกติคำสั่ง APPEND BLANK จะสร้างเรคอร์ดว่าง โดยกำหนดให้ฟิลด์ตัวอักษรมีค่าเริ่มต้นเป็นช่องว่าง ฟิลด์ตัวเลขมีค่าเป็นศูนย์ ฟิล์ดวันที่มีค่าเริ่มต้นเป็น "/ /" ส่วนฟิลด์ลอจิคัลมีค่าเริ่มต้นเป็น ".F" สำหรับเมมโมจะยังคงว่างอยู่ การป้อนข้อมูลในเรคอร์ดว่างนี้ หากป้อนไม่สำเร็จหรือยกเลิกก็จะทำให้มีเรคอร์ดว่างติดอยู่ในแฟ้ม ซึ่งถ้าต้องการลบออกโดยใช้คำสั่ง DELETE และ PACK ก็ยิ่งไปกันใหญ่

ข้อสำคัญของการทำงานที่นักเขียนโปรแกรมพึงระมัดระวังไว้เสมอคือ เมื่อต้องการเพิ่มข้อมูลให้ใช้ตัวแบ่งในหน่วยความจำรองรับข้อมูลไว้ก่อน ข้อมูลที่อยู่ในหน่วยความจำมีข้อดีคือจะยกเลิกเมื่อไรก็ได้ จะแก้ไขเปลี่ยนแปลงอย่างไรจนเป็นที่พอใจโดยไม่เกี่ยวข้องกับแฟ้ม เมื่อแน่ใจแล้วจึงนำไปใส่ในแฟ้ม

กฎที่น่าสนใจคือ พยายามทำงานกับตัวแปรในหน่วยความจำให้มากที่สุด แต่เมื่อต้องการปรับปรุงข้อมูล จึงนำไปเปลี่ยนกับข้อมูลที่มีอยู่ในแฟ้ม การดำเนินการเช่นนี้จะลดปัญหาที่เกิดกับแฟ้มข้อมูลได้มาก

บัฟเฟอร์ข้อมูลขณะเขียนอ่าน

ในการเปิดแฟ้มข้อมูลด้วยคำสั่ง USE จะเป็นการเตรียมบัฟเฟอร์ในหน่วยความจำไว้สำหรับการเขียนอ่านข้อมูล พึงระลึกว่าข้อมูลที่เปิดทุกแฟ้มจะต้องมีบัฟเฟอร์อยู่บนหน่วยความจำ คราวนี้ลองดูว่าหากเราใช้แฟ้มที่มีดัชนีร่วมอยู่ด้วย แฟ้มดัชนีก็ต้องการบัฟเฟอร์อยู่ในหน่วยความจำเช่นกัน

ขนาดของบัฟเฟอร์ของแฟ้มกำหนดขนาดโดยระบบที่เราเขียนไว้ใน CONFIG.SYS โดยปกติจะมีค่าเป็นจำนวนขนาดเท่ากับคลัสเตอร์ของข้อมูล ในฟลอปปี้ดิสค์คลัสเตอร์ที่เล็กที่สุดมีขนาด 1 กิโลไบต์ แต่ฮาร์ดดิสค์โตได้ถึง 2 หรือ 3 กิโลไบต์ต่อคลัสเตอร์ ดังนั้นเมื่อมีการเรียกข้อมูลในแฟ้มที่เปิด ระบบจะอ่านข้อมูลส่วนเรคอร์ดที่อ้างอิงถึงลงมาไว้ที่บัฟเฟอร์ ข้อมูลบริเวณนั้นจะลงมาหมดตามจำนวนของบัฟเฟอร์

ส่วนของการเขียนอ่านเป็นบัฟเฟอร์นี้เป็นเรื่องสำคัญที่ผู้เขียนโปรแกรมจะต้องคำนึงถึง เช่นเมื่อทำการเรียกข้อมูลทางเรคอร์ดมาแก้ไข เมื่อแก้ไขเสร็จแล้วข้อมูลยังคงอยู่ในบัฟเฟอร์ แต่เมื่อมีปัญหาเกิดขึ้นกับเครื่อง เช่น ไฟฟ้าดับ ข้อมูลในส่วนของบัฟเฟอร์จะเสียหาย ข้อมูลส่วนนี้อาจยังไม่ได้นำกลับเข้าไปเขียนไว้ในดิสค์จริง ดังนั้นข้อมูลที่ผู้ปฏิบัติงานคิดว่าแก้ไขแล้วจึงยังไม่แก้ไข อาจเกิดความเข้าใจผิดเป็นผลเสียกับการทำงานได้

นอกจากนี้สิ่งที่พบเห็นอีกประการหนึ่งคือ การปรับปรุงข้อมูลบางส่วนแต่ไม่ได้เขียนกลับลงในดิสค์อาจทำให้ตัวเชื่อมข้อมูลในระบบแฟ้มสัมพันธ์ (relational) เสียหายทำให้ฐานข้อมูลเกิดปัญหาได้

ในการปฏิบัติงานของผู้เขียนโปรแกรมในส่วนที่ต้องการความมั่นใจในการแก้ไขข้อมูลในส่วนของข้อมูลที่อยู่ในแฟ้มจริง จะกระทำโดยทำขั้นตอนดังนี้

- เปิดแฟ้มข้อมูลที่ต้องการแก้ไขพร้อมดัชนี
- คัดลอกข้อมูลในส่วนฟิลด์ที่ต้องการแก้ไขมาไว้ในตัวแปรหน่วยความจำ
- แก้ไขข้อมูลในส่วนตัวแปรให้เรียบร้อย
- คัดลอกจากตัวแปรในหน่วยความจำไปเก็บในแฟ้มข้อมูลในเรคอร์ดที่กำหนด
- ปิดแฟ้มข้อมูลหรือเลื่อนตัวชี้เรคอร์ดไปที่อื่นจนคาดได้ว่ามีการเขียนข้อมูลลงดิสค์แล้ว
- แสดงสถานะว่าข้อมูลได้ปรับปรุงแก้ไขแล้ว

การปฏิบัติงานกับบัฟเฟอร์นี้เป็นเรื่องที่ต้องระมัดระวัง ทุกครั้งที่เครื่องมีปัญหาเช่น เครื่องแฮง (hang) ข้อมูลในบัฟเฟอร์ยังไม่ได้ปรับปรุง จะทำให้แฟ้มที่อยู่ในดิสค์เสียหายได้

การจัดการตัวเชื่อมโยงระหว่างแฟ้ม

ลองพิจารณาแฟ้มข้อมูลที่มีความสัมพันธ์กัน โดยมีแฟ้มหลัก (parent) และแฟ้มลูก (child) ความสัมพันธ์นี้เชื่อมโยงกันด้วยข้อมูลที่มีการกำหนดในฟิลด์ เช่น ID หรือรหัส การเชื่อมโยงเหล่านี้จะเกี่ยวพันกันอยู่ เช่น ข้อมูลพนักงาน กับข้อมูลประสบการณ์การทำงาน จะเห็นว่าแฟ้มหลักคือ ข้อมูลพนักงาน แฟ้มลูกคือ ข้อมูลประสบการณ์การทำงาน โดยใช้รหัสพนักงานเป็นตัวเชื่อมโยง ดังนั้นเมื่อลบพนักงานออกจะเห็นว่าข้อมูลในแฟ้มลูกที่เชื่อมโยงอยู่จะไม่มีความหมาย ดังนั้นจึงต้องลบข้อมูลในแฟ้มลูกในส่วนของเรคอร์ดที่เกี่ยวข้องหรือสัมพันธ์กันออกไปด้วย

หากมีการเชื่อมโยงกันหลายขั้นตอนการลบจะยุ่งยากตาม ผู้เขียนโปรแกรมจะต้องเข้าใจและหาทางโยงข้อมูลทั้งหมดให้ได้ ข้อมูลที่ติดค้างอยู่โดยไม่มีการเข้าถึงของข้อมูลจะเสมือนขยะซึ่งไม่เกิดประโยชน์ การลบข้อมูลจึงเป็นเรื่องที่สำคัญหากลบคีย์เมื่อไร การค้นหาในส่วนคีย์ก็จะทำไม่ได้ การเชื่อมโยงข้อมูลตัวต่อไปก็จะไม่เกิดประโยชน์

ข้อพึงระวังที่สำคัญอีกเรื่องหนึ่งคือ การทำดัชนี การลบข้อมูลในหลายแฟ้มจะมีผลต่อโครงสร้างดัชนีในหลายตัวที่เกี่ยวข้องด้วย ผู้พัฒนาโปรแกรมต้องเน้นให้ความสำคัญในเรื่องนี้ การที่ดัชนีส่วนใดส่วนหนึ่งของระบบเสียไปจะทำให้ข้อมูลทั้งฐานข้อมูลเสียหายได้ การหลีกเลี่ยงการใช้คำสั่ง PACK และใช้การเพิ่มข้อมูลในช่องส่วนของข้อมูลที่ลบแล้วเป็นวิธีที่ทำให้ไม่เกิดการขยับข้อมูลส่วนใหญ่ในฮาร์ดดิสค์

ฉะนั้นเทคนิคการจัดการความสัมพันธ์ระหว่างแฟ้มเป็นเรื่องเทคนิคที่ต้องให้ความสนใจเป็นพิเศษ

ส่งท้าย

การสร้างฐานข้อมูลที่ประกอบด้วยแฟ้มหลาย ๆ แฟ้ม พึงระลึกว่าให้สร้างโครงสร้างแฟ้มข้อมูลที่เรียบง่าย ลดความซับซ้อนในการเชื่อมโยงกัน ลดขนาดของแฟ้มที่จะเก็บให้มีขนาดจำนวนเรคอร์ดหรือขนาดที่เก็บในพื้นที่ดิสค์ใหญ่เกินไป เน้นให้แฟ้มที่เก็บมีการเชื่อมโยงกันโดยใช้รหัส จะต้องเข้าใจว่าการจัดการข้อมูลบนไมโครคอมพิวเตอร์ยังต้องใช้เอ็มเอสดอสดูแลเรื่องการเขียนอ่านข้อมูลในดิสค์ ข้อจำกัดที่เกิดขึ้นจากดอสยังมีอีกหลายอย่าง หลีกเลี่ยงการใช้คำสั่ง PACK และ COPY เพราะจะมีการเขียนและอ่านข้อมูลครั้งละมาก ๆ